Streaming Scenes to MPEG-4 Video-Enabled Devices

Yuval Noimark
IBM Haifa Research Lab, Israel

Daniel Cohen-Or
Tel-Aviv University, Israel

We stream real-time computer graphic scenes to handheld devices using object-based encoding from MPEG-4 and knowledge from a 3D model.

Streaming a remote walkthrough of a complex computer-generated environment is an emerging challenge in computer graphics. The size of most models precludes downloading the entire model from the server to the client. Even a dynamically downloaded portion of the model determined by the client position may consist of a prohibitive number of polygons or textures. Despite the increasing availability of the Web and advances in graphics hardware, network bandwidth is a critical bottleneck.

Four common techniques for working around bandwidth bottlenecks are to

- simplify the representation of the objects to accelerate the walkthrough,[1,2]
- use a Web-based remote walkthrough,[3]
- use hybrid-rendering methods, where the client and the server both participate in the rendering of the frames,[4-6] and
- use streaming technologies to reduce the bandwidth requirement.

These approaches, however, require some computation power and programming capabilities from a client. Handheld devices such as PDAs or cell phones have limited computation power and a cell phone's programming capabilities are incomplete. To work around these limitations many device manufacturers now embed dedicated hardware for decoding MPEG-4 encoded video and audio.

Exploiting a thin client's embedded ability to decode video sequences independent of its computational power, we provide a walkthrough experience with no additional processing cost on the client and requiring no additional programming capabilities.[7] Our solution is a server-based system that streams computer-generated video rendered from an interactive remote walkthrough. The server renders the sequence—generated from the 3D model—and encodes the frames into a video stream. The stream is received by the client, which displays the stream using a video decoding chip.

System overview

Figure 1 illustrates our server-based system. The server holds the 3D model of the scene. Once interaction starts, the interactive rendering engine renders the frames, providing the scene information that the macroblock analysis module uses to generate layering information and motion hints. The video encoder uses the layering information to segment the video into foreground and background objects and to vary the quantization level locally over the frame. The motion hints accellerate encoding.

MPEG-4 encoding scheme

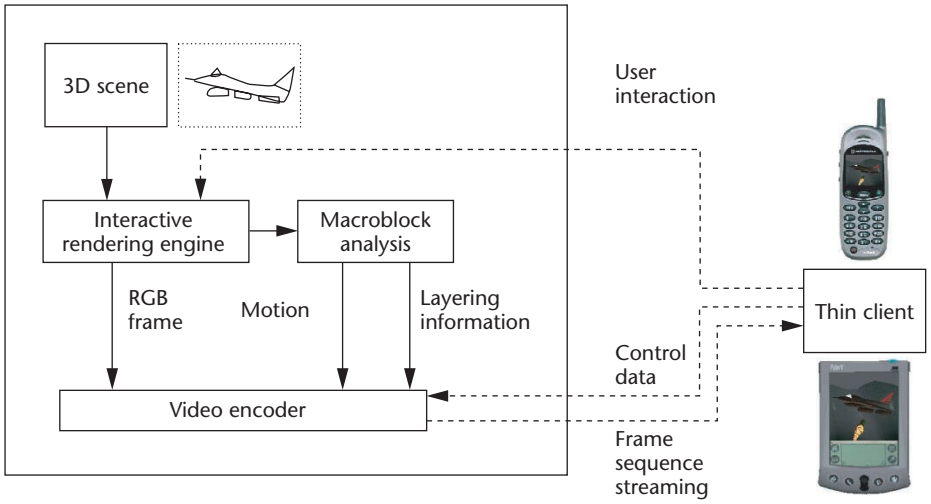
MPEG-4 is an ISO/IEC standard for compressing and representing multimedia content developed by the Moving Picture Experts Group (MPEG). Its format designation is ISO/IEC 14496.[8] An MPEG-4 encoder translates multimedia scenes into objects such as natural video, natural audio, 3D graphics, images, text, and so on. MPEG-4 compresses each object using techniques tailored to its type. A decoder reconstructs the multimedia scene from these objects.

The standard defines media objects as the primitives in the scene. Media objects can be of synthetic or natural origin and can be visual or aural. MPEG-4 part 2[9] defines the coding of the following visual objects: video, still texture, face and body animation, 2D mesh, and 3D mesh. The standard also defines how to multiplex and synchronize these objects to enable their transportation over the network.

While the 3D mesh object definition provides the tools for compressing 3D computer-generated scenes, decoding them is too complex for mobile devices. Many mobile devices can, however, decode at least one video object. Some can even decode and compose two or more objects into a single scene (for example, a background video of scenery and a foreground object of a moving vehicle).

We use the MPEG-4 video standard[10] because many

 Published by the IEEE Computer Society



1 The system overview.

mobile devices can display MPEG-4 video content using on-chip decoders.[11] The technique introduced here, however, isn't limited to this standard and could be applied to other block-based video compression schemes.

Frame coding scheme

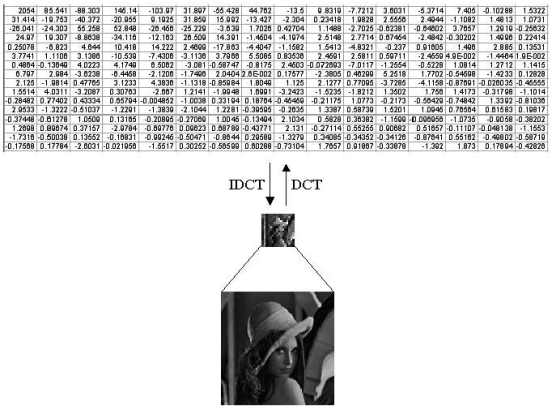
An MPEG-4 video stream consists of three types of frames: intra (I), predictive (P), and bidirectional predictive (B). MPEG-4 encodes the I frames independently of the other frames in the sequence. It encodes P and B frames using motion estimation and interpolation techniques that let them be substantially smaller than I frames. MPEG-4 uses preceding P or I frames to estimate the motion of P frames. It uses either or both preceding and successive P or I frames to estimate the motion of B frames. The standard refers to P or I frames used for motion estimation as reference frames.

More specifically, MPEG-4 encodes a frame at the macroblock level. It partitions each frame into macroblocks of 16×16 pixels and encodes them independently of the other macroblocks. The standard defines two kinds of macroblocks: intra and inter. Intra-macroblock (I-MB) data is fully encoded. Like with P and B frames, MPEG-4 encodes inter-macroblock (P-MB or B-MB) data using motion estimation.

MPEG-4 applies a discrete cosine transform (DCT) to I-MB data, quantizing the resulting coefficients and using variable length coding to encode them. Figure 2 shows an example of a DCT applied to an 16×16 frame.

For P-MBs, MPEG-4 first estimates motion by searching the reference frame for the best match for the macroblock to be encoded. The colocated macroblock is the macroblock in the reference frame, which has the same relative location as the P-MB to be encoded. MPEG-4 searches for the best match in an area around the colocated macroblock. Once it finds a motion vector, it applies a DCT to the difference between the matching area and the P-MB. It quantizes the resulting coefficients and encodes them using variable length coding.

The quantization level determines the compression ratio and controls the degree of the lossy compression. Clearly there's a tradeoff between the qualities of the



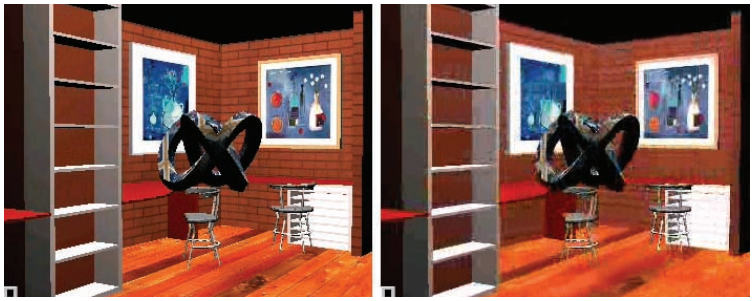
2 We scaled this image to 16×16 and used a discrete cosine transform (DCT) to represent the image in the frequency domain. We have reconstructed the image using an inverse discrete cosine transform (IDCT).

macroblocks and the degree of compression. The more the macroblock is compressed (that is, higher quantizer), the more its quality degrades. Usually an MPEG-4 encoder defines frame types using a predefined repeating sequence of I, P, B frames, like IBBPBBP or IPPBIP. During encoding of a P frame, it may, however, define each macroblock as an I-MB or P-MB independently of the global definition of the entire frame. Identifying different types of encoding per macroblock imposes an overhead of determining the estimated error. When the estimation error is too large, MPEG-4 uses an I-MB rather than prediction.

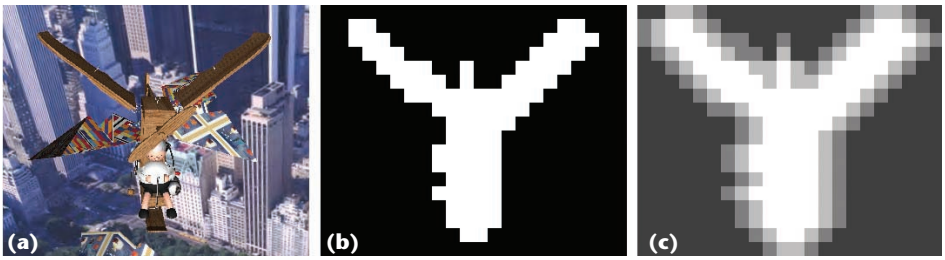
Encoding a P (or B) macroblock imposes a search for the best candidate macroblock in the reference frame. A benefit of MPEG-4 is that it doesn't restrict the search area. Searching for a candidate, however, is a complex and time-consuming task. Later we'll describe a technique we use to reduce the computation complexity of the search to enable real-time encoding.

Macroblock quantization

An important parameter in the encoding process is the quantization value. MPEG-4 quantizes and encodes the DCT coefficients into a variable size length of 3 to 26 bits. A successful choice of the quantizer is critical for an efficient encoding. A programmer usually defines



3 The effect of applying different quantization values. The quantization value for the left figure is 2, and 18 for the right figure.



4 (a) A frame from the studio sequence as created by the rendering engine. (b) The foreground/background layering information given by the macroblock analysis module. (c) Smoothing of the layering to reduce the artifacts caused by large quantizer differences.

the quantization value globally, aiming at generating some given quality according to the trade-off between quality and bit rate; the higher the quantizer the better compression achieved and the more degrading quality. The quantization factor varies from 1 to 31 in steps of 1. A step change in the quantizer can improve the compression ratio while the degradation in quality is almost unnoticeable.

MPEG-4 defines two types of quantization. In the first method, MPEG-4 uses the same quantization value for all DCT coefficients of the macroblock. In the second, it uses a matrix of quantization values for the macroblock where each coefficient can be quantized with a different value according to the matrix. In this article, we refer to the quantization method in which one quantizer is used for all coefficients.

The choice of a quantizer and a quantization method is extremely important. Quantizing the correct coefficients with the correct value will give the best compression ratios while maintaining quality as far as possible. The quantizer is also used for bit-rate control purposes. A bit-rate mechanism usually modifies the quantizer on a per-frame base. Current approaches use the human visual system to modify the quantizer on a per macroblock base.[10]

Figure 3 shows the effect of different quantization values. We encoded the left figure with a quantizer value of 2. In the figure on the right, where we use a quantization value of 18, the quality degrades significantly.

Per-layer quantization

Choosing the correct quantization value per macroblock according to its visual importance makes frame encoding more efficient. This requires some knowledge of the scene represented in the sequence.

Using depth information, or some other knowledge about the model in an image, an encoder can segment a frame into layers and assign a different visual importance value to each layer. Doing this automatically for a video stream isn't simple.[12] Real-time segmentation of video sequences isn't feasible yet.

When the frame sequence encodes views of a computer-generated model, however, the layering problem is trivial. For example, we can easily segment the image into background and foreground macroblocks as Figure 4 shows. The figure shows an image segmented and classified into two layers (see middle of Figure. 4). We colored the macroblocks containing the foreground objects in white; the less visually important macroblocks are in black. To achieve better compression ratios, we based our server system on the ability to properly select different quantization values on a per macroblock level.

While the quantization value for the background and foreground can differ by several levels—for example, we can assign a quantization value of 10 to the background and a quantization value of 2 to the foreground—such an approach can cause artifacts. The transition from foreground to background might be noticeable. Our approach uses a gradual change of the quantization value, so that the transition from foreground to background isn't abrupt and noticeable (see Figure 4).

Some methods that assign different quantization values on a per macroblock basis don't consider regions of interest.[13] We base our approach on the knowledge of the model and on the visual importance of its different parts. We compress regions that are less visually important with a higher ratio while preserving the appropriate quality for regions of higher importance to the viewer.

Real-time encoding

To provide real-time interactivity on the client, we must generate the video in real time. This imposes a huge requirement from the server, which needs to both render and encode in real time. Exploiting the information in the 3D scene to speed up selection of the encoding pattern and accelerate motion estimation, we reduce the computation power required from the server and enable a real-time experience.

Selecting the encoding pattern

Selecting an I frame when we know we need it further exploits the computer-generated model to compress the frame sequence. In the encoding process, I frames (except for the first frame) seek capability—playing a video sequence from a new location—and reduce the propagated error of P frames over error-prone networks.

A conventional encoder uses a predefined pattern to determine the type of frame to encode. Patterns, such as

IPPPP or IPBBBPBBI determine which frame should be independently encoded and which should be encoded using motion estimation. Most of the frame types—as defined in these encoding patterns—are motion estimated.

In the case of scene cuts, where the whole scene changes in a given frame, no motion vectors exist. The motion estimation error is too high. Encoding a scene cut as a P frame imposes a redundant search for each of its macroblocks.

We can use the scene's model to determine when I frames are necessary. The encoding pattern isn't predefined but determined in real time as a function of the walkthrough. Our system detects scene cuts and informs the frame encoder to avoid the search for motion and encode the current frame as an I frame. This considerably reduces the frame encoder's complexity, diverting processing power to other time-consuming tasks.

Our system also informs the frame encoder to encode a frame as an I frame when the network is error prone. The client transmits statistics of lost frames to the server when a back channel is available. When the number of lost frames increases, the server increases the number of I frames. This reduces the propagated error of P frames.

Accelerating motion estimation

Searching for a motion vector is the most time-consuming task in the encoding. While other researchers have proposed faster search algorithms, there's typically a direct relation between the complexity of the search algorithm and quality of the match.[10] An exhaustive search yields the best possible match, but is, of course, overly expensive. Algorithms with relatively low complexity can provide acceptable results for real-time applications. Wallach et al. introduced a technique to find motion vectors based on optical flow information available using common graphics hardware.[14]

We use a logarithmic motion estimation algorithm in our implementation—a hierarchical search that checks the corners, sides, and centers of a $2N \times 2N$ pixel square, for the best match and continues recursively using the best result as the new center. Each iteration (four corners, four sides, one center) has nine possible motion vectors.

The macroblock analysis module exploits the known model to accelerate the search algorithm by providing approximate motion vectors to the encoder module. Based on the available scene model and the available camera parameters, it computes an approximate optical flow for a given macroblock. The encoder uses this as the initial location of a logarithmic search. Saving the first level search eliminates nine searches (a single iteration of the logarithmic search algorithm), speeding encoding and yielding a real-time encoding throughput. The speedup gained in the logarithmic search time comes with a cost: computing the optical flow is also time-consuming. Because the macroblock analysis and the frame encoder work simultaneously, in a pipeline fashion, the overhead is hidden in our system. This acceleration of the encoding improves the system's overall performance and enables its use in real-time scenarios.

5 Composition of a rotating geometric shape on a studio background.

Object segmentation

MPEG-4 introduces the representation of media as a composition of objects. Using that scheme, MPEG-4 achieves higher compression ratios than MPEG-1 or MPEG-2 and stores the media efficiently by sharing common objects. For example, a foreground object of a news anchor and a background object of the image behind him can represent a news anchor video. Figure 5 illustrates this with a rotating geometric shape on a studio background.

Video cameras capture each frame as a whole, without any relation to the objects composing it. Video can, however, be captured as objects. For example, a television studio can capture a news anchor and his background independently using a blue screen, creating foreground and background objects. Still, most of the video content isn't represented as objects.

While difficult to perform for video streams, object segmentation is trivial for 3D scenes. The geometry representation is already object based. Because our system encodes 3D scenes as video, we can segment it at no cost. Moreover, the almost constant motion of the background in a typical walkthrough scene further increases the number of frames per second that can be encoded in real time.

While MPEG-4 decoders on current handheld devices do not support object composition, a few in development do, and we anticipate most future devices will. Using it in our server allows us to take advantage of sprite coding, improving the compression ratio for our video and speed video encoding.

Table 1. Comparison between the encoding time in seconds of the foreground only and of the whole frame.

	Quarter Common Intermediate Format		Common Intermediate Format	
Encoding Time of	Studio	Pilot	Studio	Pilot
Foreground only	0.9	1.1	2.8	3.7
Whole frame	2.5	2.4	10.8	10.2

Table 2. The stream sizes of Common Intermediate Format (352 × 288 frame) files in different quantization levels.

	Studio Sequence			Pilot Sequence		
Quantization	High (Kbytes)	Two levels (Kbytes)	Low (Kbytes)	High (Kbytes)	Two levels (Kbytes)	Low (Kbytes)
8-2	1,813	3,037	7,307	1,657	3,404	6,947
10-4	1,388	1,980	3,851	1,249	2,121	3,596
12-6	1,105	1,459	2,516	979	1,514	2,326
14-8	908	1,143	1,813	788	1,151	1,657
16-10	767	932	1,388	652	912	1,249
18-12	660	783	1,105	548	744	979

Representing a large background

MPEG-4 defines sprite coding, which represents large backgrounds as a mosaic. The client reconstructs only the portion relevant to the frame being displayed. The client can compose arbitrary shape video over the background, resulting in an efficient representation of the whole video sequence.

At the beginning of a transmission, the server transmits the whole sprite. For each frame in the sequence, however, it need only transmit the parameters indicating the portion of the sprite to be displayed. A server may also transmit a sprite in several parts so that the whole background is built up progressively.

A sprite contains global motion information using eight parameters that describe the affine transform of the sprite between sequential frames. The client uses these parameters to warp the sprite for each frame. This is known as global motion compensation (GMC).

Exploiting the GMC technique, we segment the background as we render the 3D scene. We render each frame without the foreground objects. We transmit the resulting background image as a sprite object then transmit only the updated parts of the sprite. Our system generates the GMC information for the client from the viewing matrices (translation and rotation) of the 3D scene. In cases where the movement of the camera can't be represented as affine transform, such as zoom in/out, we update the whole sprite.

Foreground and background objects

We transmit the foreground and background as separate objects. We transmit the foreground object as a nonrectangular video. For the background, we first compute the global motion estimation (GME) using information on the scene and changes in camera viewpoint, then transmit (stream) the background along its GMC parameters.

The mobile device—capable of decoding and composing MPEG-4 video objects—decodes and composes the foreground and background, reconstructing the original scene. With this approach, we improve the compression ratio for the whole video (mostly due to the GMC technique applied to the background).

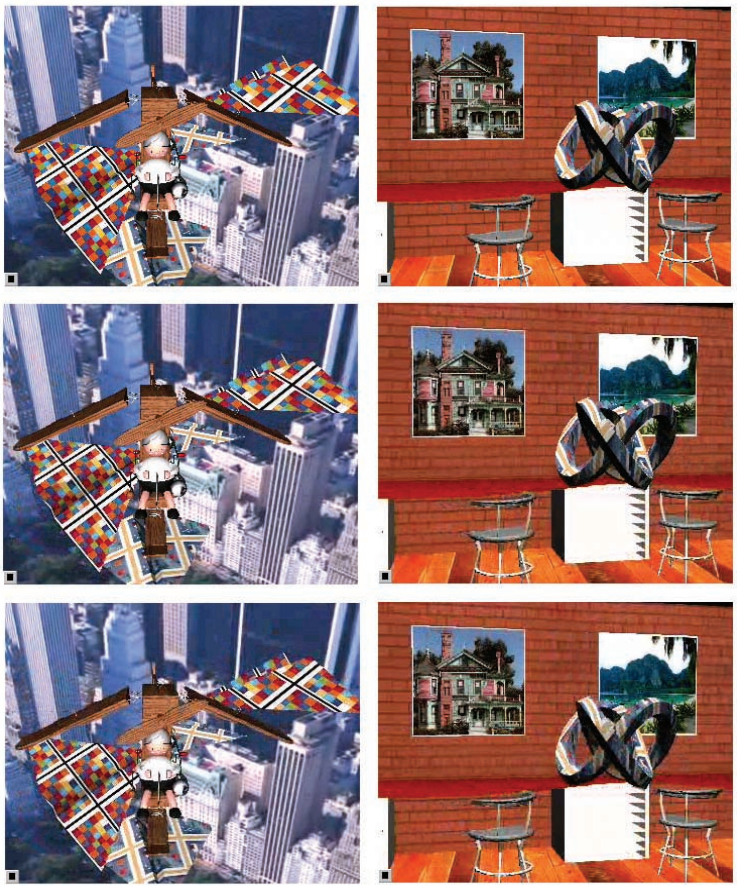
Leveraging knowledge of the scene to encode only the foreground object, we perform real-time segmentation without loading the system with the processing power required for traditional object segmentation approaches. Table 1 shows that our approach is much more efficient in terms of computation power.

Results

Implementing our technique within the framework of MPEG-4, we wrote the encoder in C while the decoder is pure Java. While the rendering engine is based on a proprietary implementation, the decoder can be used on any Java-enabled platform. To guage the effectiveness of our technique we compared the stream size created by our two-level quantization to conventional constant quantization for two kinds of sequences generated using Enbaya. The studio sequence has global motion that occurs in walkthroughs when turning. The pilot sequence is more static and mainly includes motion of the object itself.

Table 2 compares the stream size of the frame sequences encoded with a constant quantization value and those encoded with our method. The "Quantization" column shows the quantization range used (high to low). The lower value represents foreground quantization and the higher value represents background quantization. The "Two levels" column shows our resulting stream sizes that are lower than the sizes of the streams in the "Low" columns. The "High" column shows the sizes of streams encoded with the same value we used for the background. Although its size is lower, its quality is worse than in our two-levels approach. This is especially noticeable in the areas of the foreground object.

Figure 6 compares images encoded by our technique and naive quantization, in which we used the same quantization value for all macroblocks. We encoded the two sequences with different ranges. The left column is from the "Pilot" sequence, encoded in the range of 10-4. The right column is from the "Studio" sequence, encoded in the range of 12-6. In the upper row, we used the lowest value as the constant quantization value. In the lowest row, we used the higher value. The middle row shows frames of our two-layer technique. Note that the quality of the foreground object is as high as that in the upper row.



6 A comparison of images encoded by our technique and naive quantization.

Figure 6 shows that our method provides better compression ratios while only degrading the quality of the background regions. We use the same quantization value for the foreground as in the constant quantization and a higher quantization value for the background. The background regions have less importance to the viewer and therefore the degradation in quality is acceptable.

The size of each frame in the sequence for Table 2 is Common Intermediate Format (CIF) (352 × 288). We repeated the same analysis for a Quarter Common Inermediate Format (QCIF) (176 × 144) size type, as Table 3 shows. The results are similar, however the compression gained is somewhat less than in the CIF size. This is because the macroblock size is fixed and therefore the ratio between the foreground and the background macroblocks is a bit higher for QCIF.

Conclusions and future work

We introduced a server-based system that alleviates the processing power required by a thin client to render an interactive remote walkthrough. The technique exploits the fact that a thin client has the ability to decode video sequences independently of its computational power. Because we're streaming computer-generated video, we benefit from the available model of the scene to better compress the sequence and reduce the required bandwidth and communication latency.

While current bandwidth issues might delay the adoption of this technique, we expect it will be adopted as emerging networks gain more bandwidth. It doesn't, however, require broadband networks. We can apply this technique in interactive network games where the server needs to serve several clients playing in a common virtual environment.

The method we presented in this article focuses on the basic set of MPEG-4 video. MPEG-4 defines various types of other media objects rather than video objects. Today's devices, however, are still limited to MPEG-4 video decoding and don't provide the ability to display a whole MPEG-4 scene. Moreover, the devices can't compose several overlaid video objects. Only a few devices

Table 3. The stream sizes of QCIF (176 × 144 frame) files in different quantization levels.

	Studio Sequence			Pilot Sequence		
Quantization	High (Kbytes)	Two levels (Kbytes)	Low (Kbytes)	High (Kbytes)	Two levels (Kbytes)	Low (Kbytes)
8-2	512	1,160	2,257	430	1,190	1,788
10-4	378	699	1,158	319	698	938
12-6	291	485	735	244	476	609
14-8	230	359	512	191	347	430
16-10	188	278	378	152	264	319
18-12	157	223	485	124	207	244

can support the foreground and background object segmentation technique we present here. As handheld devices become more powerful and can support more MPEG-4 features, we can enhance this technique and further improve its performance. ■

Acknowledgment

We thank the following people for their various contributions: Efi Fogel and Amit Shaked from Enbaya Ltd., Zohar Sivan and Konstantin Kupeev from IBM Research Lab in Haifa.

References

1. T.A. Funkhouser and C. H. Sèquin, "Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments," *Computer Graphics Proc.* (Annual Conf. Series), ACM Press, 1993, pp. 247-254.
2. D.G. Aliaga and A.A. Lastra, "Architectural Walkthroughs Using Portal Textures," *Proc. IEEE Visualization 97*, R. Yagel and H. Hagen, eds., IEEE CS Press, 1997, pp. 355-362.
3. E. Teler and D. Lischinski, "Streaming of Complex 3D Scenes for Remote Walkthroughs," *Computer Graphics Forum* (Proc. Eurographics 2001), Blackwell Publishers, 2001, pp. 17-25
4. Y. Mann and D. Cohen-Or, "Selective Pixel Transmission for Navigating in Remote Virtual Environments," *Computer Graphics Forum*, vol. 16, no. 3, Aug. 1997, pp. 201-206.
5. I. Yoon and U. Neumann, "Web-Based Remote Rendering with IBRAC (Image-Based Rendering Acceleration and Compression)" *Computer Graphics Forum*, vol. 19, no. 3, Aug. 2000, pp. 321-330.
6. D. Cohen-Or et al., "Deep Compression for Streaming Texture Intensive Animations," *Computer Graphics Proc.* (Annual Conf. Series), ACM Press, 1999, pp. 261-268.
7. D. Cohen-Or, Y. Noimark, and T. Zvi, "A Server-Based Interactive Remote Walkthrough," *Proc. 6th Eurographics Workshop on Multimedia*, Springer-Verlag, 2001, pp. 75-86.
8. R. Koenen, "Overview of the MPEG-4 Standard (ISO/IECJTC1/SC29/WG11 N3747)," http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm, 2000.
9. *ISO/IEC 14496-2, Information Technology—Coding of Audio-Visual Objects—Part 2: Visual*, Int'l Organization for Standardization, Geneva, 2001.
10. A.M. Tourapis, O.C. Au, and M.L. Liou, "Fast Motion Estimation Using Circular Zonal Search," *Proc. SPIE Symp. Visual Comm. and Image Processing* (VCIP 99), vol. 2, SPIE Press, 1999, pp. 1496-1504.
11. M. Budagavi et al., "Wireless MPEG-4 Video on Texas Instruments DSP Chips," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, vol. 4, IEEE Computer Society, 1999, pp. 2223-2226.
12. H. Huang et al., "New Video Object Segmentation Technique Based on Flow-Thread Features for MPEG-4 and Multimedia Systems," *Proc. Image and Video Comm. and Processing*, SPIE Press, 2000, pp. 204-212.
13. H.-J. Lee, T. Chiang, and Y.-Q. Zhang, "Scalable Rate Control for Very Low Bit Rate (VLBR) Video," *Proc. 1997 Int'l Conf. Image Processing* (ICIP 97), vol. 3, IEEE CS Press, 1997, pp. 768-772.
14. D.S. Wallach et al., "Accelerated MPEG Compression of Dynamic Polygonal Scenes," *Proc. Siggraph*, ACM Press, 1994, pp. 193-197.

Yuval Noimark *is a research staff member of the IBM Research Lab in Haifa, Israel. He is the technical lead of MPEG-4 activity. He is a graduate student in the School of Computer Science at Tel-Aviv University, Israel. His research interests include video compression, computer graphics, Java multimedias and multimedia frameworks. He has a BSc cum laude in computer science from the Technion, Israel Institute of Technology*

Daniel Cohen-Or *is an associate professor at the School of Computer Science at Tel-Aviv University. His research interests include computer graphics, rendering techniques, client–server 3D-graphics applications, real-time walkthroughs and flythroughs, 3D compression, visibility, meshes, and volume graphics. He has a BS cum laude in both mathematics and computer science and an MSc in computer science from Ben-Gurion University, and a PhD from the Department of Computer Science at the State University of New York at Stony Brook.*

Readers may contact Yuval Noimark at the IBM Haifa Research Lab, Haifa 31905, Israel, email noimark@il.ibm.com.